Parallel Bayesian Network Structure Learning with Application to Gene Networks

Olga Nikolova
Bioinformatics and Computational Biology
Department of Electrical and Computer Engineering
Iowa State University, Ames, Iowa, USA
Email: olia@iastate.edu

Srinivas Aluru
Department of Electrical and Computer Engineering
Iowa State University, Ames, Iowa, USA
Department of Computer Science and Engineering
Indian Institute of Technology Bombay, Mumbai, India
Email: aluru@iastate.edu

***Abstract*—Bayesian networks (BN) are probabilistic graphical models which are widely utilized in various research areas, including modeling complex biological interactions in the cell. Learning the structure of a BN is an NP-hard problem and exact solutions are limited to a few tens of variables. In this work, we present a parallel BN structure learning algorithm that combines principles of both heuristic and exact approaches and facilitates learning of larger networks. We demonstrate the applicability of our approach by an implementation on a Cray AMD cluster, and present experimental results for the problem of inferring gene networks. Our approach is work-optimal and achieves nearly perfect scaling.**

I. Introduction

Bayesian networks (BNs) are probabilistic graphical models which enable intuitive and holistic representation of complex interactions between variables in a given domain. The structure of a BN is learned from data and is commonly used to understand the mechanisms of interaction, reason about causal and cascade effects, predict outcomes of system perturbations, etc. BNs are applicable in many research areas including medical and fault diagnosis, automated vision, robotics, information retrieval, and bioinformatics.

In recent years, BNs have been applied in systems biology to uncover genetic regulatory relationships [9], [20]. In this problem, genes are modeled as random variables which correspond to nodes in a network, and directed edges represent interactions. Gene expression data measured under different experimental conditions are used to learn the most probable graphical representation of the dependencies among the analyzed genes. Gene expression measurement techniques such as microarrays, and the newer high-throughput sequencing technologies, have led to the accumulation of large public repositories of gene expression data. Each experiment provides the measure of gene activity on a whole genome scale, typically including all experimentally measurable expressed elements in a given organism. However, learning the structure of a BN is computationally demanding, and inferring large networks remains a daunting problem.

A. Previous Work

Learning of BNs and their applications in many fields have been well-studied, and several exact and heuristic approaches for structure learning of BNs have been proposed. Exact structure learning of BNs was shown to be NP-hard even for the instance of the problem when the node in-degree is bounded [7]. Exact approaches explore the super-exponential search space of all possible directed acyclic graphs (DAG) over a given set of variables [19]. While in recent works [17] the run-time has been reduced to exponential, the memory requirement also grows exponentially in the number of variables. This is often a serious bottleneck, even for modest problem sizes involving few tens of variables. This has prompted the development of heuristic approaches, which remains an active area of research.

Most heuristic approaches for BN structure learning can be characterized as either constraint-based, or based on score optimization, or hybrid approaches that include aspects of both. In constraint-based approaches, conditional independencies between variables are evaluated using statistical tests (G^2, partial correlation, etc.) and propagated to infer a partially oriented network. Then, axioms are used to orient the remaining edges such that all constraints are satisfied [6], [22]. These methods have produced mixed results, due largely to the sensitivity of the statistical tests to noise in the data.

Score optimization methods optimize a scoring criterion while exploring the search space [8], [10], [13], [24]. The scoring criterion evaluates how well a proposed graph predicts dependencies in the observed data. Commonly, a classic hill-climbing technique over the search-space of DAGs is employed. At each step, every possible edge addition, removal, and reversal is evaluated under the acyclicity constraint, and a step is taken towards the highest scoring change. These steps are repeated until the overall network score can no longer be improved. This classic idea has been enhanced by adding TABU lists and random restarts to avoid local optima. A novel operator has also been proposed, which allows for the reinsertion of a particular node in a new location in the graph [13]. Variations of this stream of approaches over DAGs have been proposed, where considerations over orders are made [8], [24]. These methods explore reductions in search space imposed by a topological sort on a DAG. Alternatively, relaxing the acyclicity constraint has been proposed, which allows for the computation of optimal set of neighboring nodes (both incoming and outgoing edges). Then, edges are removed to break cycles and to eventually

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

restore the acyclicity constraint, such that the overall network score is optimized [18]. Reductions in search space have been proposed by pre-selecting subsets of variables for each node. One way to do this is to use a constraint-based approach and eliminate edges between conditionally independent variables. Such methods are sometimes referred to as hybrid methods [25]. This category of approaches is generally considered a promising direction.

In stark contrast to the vast literature on serial algorithms for BN structure learning, and despite the compute and memory intensive nature of the problem, there is limited work in the development of parallel methods for structure learning. In 2009, Nikolova *et al.* [16] developed the first parallel algorithm for exact BN structure learning that is both work and space optimal. While this work allows pushing the limits on the scale of networks that can be inferred with precision, the NP-hard nature of the problem limits the applicability of this solution to small networks of 30-40 nodes. Heuristics based approaches that trade off optimality for the ability to learn larger networks are the only means to structure learning of large-scale BNs. While many score optimization and hybrid heuristic approaches that yield good quality results have been reported, there is little work in developing efficient parallel counterparts to these algorithms. Some works have resorted to simplistic parallelization of such approaches which may adversely affect quality. For example, Tamada *et al.* [23] build whole-genome scale networks by selecting a random subset of nodes on which a serial method can be run, and exploiting parallelism by conducting many such runs with multiple random subsets. The results of these runs are subsequently combined to form the larger network. While the quality of the network improves with the number of random samples, the true parents of any given node may not be simultaneously part of any one of the subsets considered.

B. Contributions

In this paper, we present a parallel approach to build large-scale BNs that combines a heuristic framework for scaling with the precision of exact learning. In this approach, we first identify the set of candidate parents for each node in the network with the goal that the set so identified is a superset of the true parents (for subsequent correctness) and is as small as possible (for computational efficiency). Taking these sets as basis, we develop a parallel exact algorithm for collectively inferring the true parents of each node in the network. We expect that the resulting network should be of higher quality due to the utilization of the exact algorithm in refining the candidate parents sets. Nevertheless, no heuristic algorithm can guarantee optimality of the network. Sources of error in this approach include non-inclusion of actual parents in the candidate parent sets and how faithfully the scoring function models the interactions. As a result, the network derived by inferring the parent sets may contain cycles. We augment this with a parallel algorithm to detect and break cycles to result in a DAG that is returned as the inferred structure. We apply the presented approach to reconstruct gene networks of the model plant *Arabidopsis thaliana*.

II. Bayesian Networks and Structure Learning

Throughout this paper, we use indexed upper-case letters to denote random variables (e.g. X_i). Let $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ be a set of random variables. In a directed acyclic graph (DAG) on $\mathcal{X}$, a node X_j is called a *descendant* of X_i if a directed path exists from X_i to X_j, and a *nondescendent* otherwise. Further, a node X_k is called a *parent* of X_i, and X_i is called a *child* of X_k, if there is an edge from X_k to X_i. The set of all parents of X_i is denoted by $Pa(X_i)$. The following is known as the Markov assumption:

Definition 1. *Let P be a joint probability distribution over the variables in set $\mathcal{X}$ and $N = (\mathcal{X}, E)$ be a DAG. Then N and P are said to satisfy the Markov assumption if each variable $X_i \in \mathcal{X}$ is conditionally independent of the set of its nondescendants, given the set of its parents [14].*

The pair (N, P) is called a BN if N and P satisfy the Markov assumption. This allows for a compact representation of the joint probability over $\mathcal{X}$, as the product of the conditional probabilities of each variable given its parents:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid Pa(X_i)).$$

Let $D_{n\times m}$ be a data set of m observations for each of the n variables $X_i \in \mathcal{X}$. In other words, we assume no missing values. Let $Score(N : D_{n\times m})$ be a scoring function which evaluates how well a given DAG N predicts the dependencies in $D_{n\times m}$. Finding an optimal BN structure is equivalent to finding a DAG which optimizes the scoring function: $N_{opt} = \underset{N}{\operatorname{argmax}}\, Score(N : D_{n\times m})$. Since multiple DAGs could fit the data equally well, N_{opt} is a DAG from the class of BNs which best predict the data.

Various scoring functions have been proposed in the literature for this task. Among the most commonly used ones are the Bayesian Information Criterion (BIC) [21], Minimum Description Length principle (MDL) [12], the Bayesian Dirichlet (BD) [8] and its extension Bayesian Dirichlet equivalence (BDe) [11] scoring functions. A common property among these is that they are decomposable, i.e., they allow for the score of the entire network to be computed as a sum of the individual score contributions $s(X_i, Pa(X_i))$ of each variable, given its parents:

$$Score(N) = \sum_{X_i \in \mathcal{X}} s(X_i, Pa(X_i)).$$

In addition, the BDe scoring function has the *score equivalence* property, which guarantees that if two DAGs represent the data equally well, then they will be assigned the same score.

III. Problem Statement and the Proposed Approach

Given a set $\mathcal{X}$ of n variables and a fully-observed data matrix $D_{n\times m}$ we wish to find a DAG N such that it optimizes

a scoring function $Score(N)$. Our work draws upon the two stage algorithm presented by Peng *et al.* [18], which is reported to produce results comparable to or better than other large-scale BN learning tools. However, it is not a direct parallelization of this work and employs a very different strategy for the first stage in order to inject an optimal exact structure learning algorithm for improving quality, while developing a parallelization of the second stage. We briefly review Peng *et al.*'s sequential algorithm next.

A. Overview of the Sequential Approach

In this approach, structure learning is carried out by independently finding the locally optimal parents set for each node in the network. The network is inferred by combining the results from this stage for all nodes in the network. The resulting network is not necessarily a DAG and may contain cycles. In the second stage, cycles are broken until the graph becomes acyclic while the score of the overall network is maximized.

1) Finding Locally Optimal Parents: To pre-compute a set of locally-optimal parents for each variable in the network, a simple incremental routine is employed. The parents set of each node is initialized to the empty set. In each iteration, the next single best parent for a given node is added to the current parents set. The search continues until the overall score of the variable given its parents $s(X, Pa(X))$ can no longer be improved.

2) Minimal Score-loss Cycle Elimination: Precomputing locally optimal parents as described above may result in a network containing cycles. To eliminate a simple cycle, it is sufficient to remove any of the edges in the cycle. The removal of any given edge (X_i, X_j) is equivalent to removing node X_i from the parents set of X_j. The score-loss function δ_{ij} provides an estimate of the reduction in score, which may result from this change:

$$\delta_{ij} = s(X_j, Pa(X_j)) - s(X_j, Pa(X_j) \backslash \{X_i\}).$$

In simple cycles, the edge with smallest score-loss is removed. In nested cycles, removing an edge shared by multiple cycles could break more than one cycle. Thus, in the presence of nested cycles the edge shared by most cycles is first identified. Then the score-loss of this edge is compared to the sum of smallest score-loss edges in each of the participating cycles. If the score-loss resulting from the removal of the most shared edge is minimal, then this edge is removed and the remaining cycles are re-valuated in a similar manner. Otherwise, in each cycle the edge with the smallest score-loss is removed.

Short-Cycle-First Heuristic: We next comment on how and in what order cycles of different lengths are detected. Cycles in a given graph can be identified by exponentiation of its adjacency matrix. If $M_{n\times n}$ is the adjacency matrix of graph G over n variables and the trace of $M^q_{n\times n}$ is not 0, then G contains cycles of length q. Peng *et al.* [18] make the observation that when cycles of different lengths are present, breaking short cycles first is more efficient than longer ones, as this results in a higher probability that more than one cycle is eliminated this way. Thus, they identify cycles efficiently by exponentiation of the adjacency matrix from the lowest to the highest power.

After cycles are detected, they can be easily identified using depth first search. The multiplicity of each variable (the value in the main diagonal in $M^q_{n\times n}$ for each node) indicates the number of cycles in which the node participates and is used for the identification of the most shared edge in minimal score-loss cycle elimination.

IV. Overview of the Parallel Approach

While we follow the overall framework of Peng *et al.* [18], we propose a different strategy for the first stage to more accurately determine the locally optimal parents of each node. This is carried out in two steps. Let n denote the number of variables/nodes in the BN. In the first step, we compute the candidate parent set of each node in the network. The goal here is to reduce the size of each candidate parent set from the maximum possible $n-1$ to as small a size as possible, without running into the risk of omitting a legitimate parent. This can be achieved using a variety of algorithms. For instance, Nikolova *et al.* [15] present a parallel algorithm to compute candidate parent sets based on the MMPC algorithm [25]. This fits well with the proposed approach as the entire framework will then run in parallel.

In other cases, there may be domain-specific strategies for identifying candidate parent sets. For example, the specific application of interest in our work is to infer gene regulatory networks. For this problem, it is natural to identify candidate parent sets based on pairwise correlations – if two genes are co-expressed as per a correlation measure, then each gene is put in the candidate parent set of the other. Correlations can be computed using multiple measures such as Pearson correlation, Spearman correlation, etc. In this work, we utilize the more rigorous parallel information-theoretic approach proposed by Zola *et al.* [27], that uses pairwise mutual information to detect nonlinear interactions, removes indirect interactions to further reduce candidate parent set sizes, and carries out permutation testing to ascertain the validity of predicted interactions.

The parallel network structure learning algorithm proposed in this work starts off with the candidate parents sets as input, obtained through one of the means explained above or another such method. The key contribution we make is to develop a parallel exact method to determine the locally optimal parents, and a parallel method to break cycles from the resulting network so obtained. We expect that the exact algorithm, which exhaustively evaluates all subsets of each candidate parents set, will yield a superior quality network.

In case of large networks, the sizes of some of the pre-computed candidate parents sets may be too large for the exact algorithm. In case of gene regulatory networks, it is highly likely that a gene with large node degree is a regulatory element that influences many other genes (hence the parent), and it is unlikely that the expression of a gene is influenced by a large number of genes. Hence, we compute locally optimal parents sets for all nodes whose candidate parents sets do not

exceed a threshold value, determined by the capability of the exact algorithm. For a node X_i whose candidate parents set size exceeds this threshold, we remove those nodes X_j which are already evaluated by the exact algorithm and the result affirms X_i is a parent of X_j. Thus, by eliminating children of X_i from its candidate parents set, we hope to reduce the size of this set to below the threshold required for the exact method. This procedure works because children of a node are often included in the candidate parents set of a node, due to the inability to distinguish between parents and children at this stage of the algorithm. For example, a correlation measure used to measure co-expression will infer an undirected network that is unable to differentiate between the parents and children of a node.

For the second stage of cycle removal, we propose a parallel method to carry out the cycle identification procedure by matrix exponentiation in short-cycle-first manner. Identified cycles are then resolved using the minimal score-loss criterion. The subsequent sections describe in detail our parallel algorithm.

V. Parallel Discovery of Locally Optimal Parents

We first present a parallel algorithm to compute the set of locally optimal parents for a single variable. Next, we demonstrate that it is advantageous to compute sets of locally optimal parents simultaneously for multiple variables. Finally, we present a scheduling scheme to pool together multiple variables and compute their locally optimal parents sets simultaneously.

A. Optimal Parents Set for a Single Variable

Let $CP(X_i) \subset \mathcal{X}$ be the candidate parents set for the variable X_i, and $A \subseteq CP(X_i)$. Using $D_{n\times m}$, a scoring function $s(X_i, A)$ determines the score of choosing A to be the set of parents for X_i; or equivalently, $s(X_i, A)$ specifies how well A predicts X_i given $D_{n\times m}$. The precise formula for function s is not relevant in the context of this paper, beyond the fact that its computation requires $O(mn)$ time.

Definition 2. *Let $CP(X_i)$ be the candidate parents set of X_i and $A \subseteq CP(X_i)$. Let $F(X_i, A)$ denote the highest possible score that can be obtained by choosing parents of X_i from A; i.e.,*

$$F(X_i, A) = \max_{B\subseteq A} s(X_i, B).$$

A set B which maximizes the above equation is an optimal parents set for X_i ($OP(X_i)$) from the candidate parents subset A. While $F(X_i, A)$ can be computed by directly evaluating all $2^{|A|}$ subsets of A, it is advantageous to compute it based on the following recursive formulation. For any non-empty set A,

$$F(X_i, A) = \max \begin{cases} s(X_i, A) \\ \max\limits_{X_j \in A} F(X_i, A\backslash\{X_j\}) \end{cases}.$$

This formulation allows for a dynamic programming approach, which can be visualized as operating on the lattice

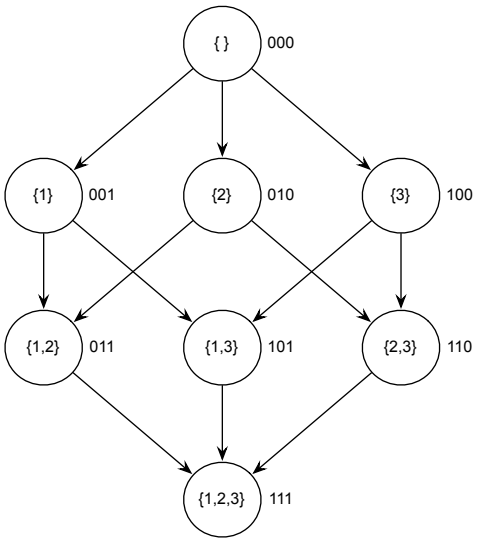

Fig. 1. A subset lattice for a candidate parents set $CP(X_i) = \{1, 2, 3\}$ of $d = 3$ variables and its partitioning into 4 levels, where set $\{\}$ is at level 0.

L formed by the partial order "set inclusion" on the power set of $CP(X_i)$. Let $|CP(X_i)| = d$. Then the lattice L is a directed graph (V_L, E_L), where $|V_L| = 2^d$ and $(B, A) \in E_L$ if $B \subset A$ and $|A| = |B| + 1$. The lattice is naturally partitioned into levels, where level $l \in [0, d]$ contains all subsets of size l (see Figure 1).

A node $A \subseteq CP(X_i)$ at level l has l incoming edges from nodes $A\backslash\{X_j\}$ for each $X_j \in A$, and $d - l$ outgoing edges to nodes $A \cup \{X_k\}$ for each $X_k \in CP(X_i)\backslash A$. Exactly one F score is computed at node A, i.e $F(X_i, A)$. The computed value is further used by the nodes incident to the outgoing edges. Each level in the lattice can be computed concurrently, with data flowing from one level to the next.

Nikolova *et al.* present an optimal parallel algorithm for the problem of exact induction of optimal BN structures from data [16]. While the memory requirements in the problem of exact BN structure learning would be prohibitive at the scale of interest in this work, the mapping strategy can be adapted for the problem of learning optimal parents sets. This is due to the significant reduction in memory requirements by 1) preselecting sets of CPs for each variable, and 2) relaxing the acyclicity constraint.

B. Run-time and Space Complexity

In [16] a similar subset lattice is efficiently mapped onto $p = 2^k$ processors, where $k < n$. Here we omit a comprehensive review of the method in [16] and instead present the differences unique to mapping the computation of a single F score. As $|PC(X_i)| = d$ we have a total of 2^d subsets to consider. The serial complexity of computing a single F score is given by

$$\sum_{i=0}^{d} (i + im) \cdot \binom{d}{i} = O\left(mn \cdot 2^d\right).$$

In parallel, we have a total of $k+2^{d-k}$ parallel steps. At each step, at least one processor computes at least one F function, which yields $O(mn)$ time. Thus, the parallel run-rime is given by $O\left(mn \cdot \left(k+2^{d-k}\right)\right)$. Given that the serial run time is $O\left(mn \cdot 2^d\right)$, optimal parallel run-time is $O\left(mn \cdot 2^{d-k}\right)$, which is achieved by our approach when $k + \log k < d$.

The total number of F-scores computed by the sequential algorithm is

$$\sum_{i=0}^{d} \binom{d}{i}.$$

Since storing only two levels at a time is sufficient, maximum number of F-scores is stored when $i = \lceil \frac{d}{2} \rceil$, which using Stirling's formula approximates to $\sqrt{\frac{2}{\pi}} \cdot \frac{1}{\sqrt{d}} \cdot 2^d$. In the parallel case, the total number of F-scores to be computed is

$$\sum_{i=0}^{d-k} \binom{d-k}{i}.$$

As before, the maximum storage needed at the same time is given by the summation of two consecutive terms in the above summation. Such terms are maximized when $i = \lceil \frac{d-k}{2} \rceil$. Then the maximum number of elements stored per processor becomes

$$\binom{d-k}{\lceil \frac{d-k}{2} \rceil},$$

or using Stirling's formula approximately $\sqrt{\frac{2}{\pi}} \cdot \frac{1}{\sqrt{d-k}} \cdot 2^{d-k}$. Thus, the ratio serial/parallel of number of stored elements yields

$$\frac{\sqrt{d-k}}{\sqrt{d}} \cdot 2^k,$$

which is optimal.

C. Simultaneous Computation of Optimal Parents Sets for Multiple Variables

1) Mapping: We next note that multiple variables may have candidate parents sets of the same size. For example, Figure 2 depicts the node degree distribution of candidate parents sets for two data sets: $n = 250$ and $m = 200$, and $n = 500$ and $m = 100$. During the computation of any two OP sets, the communication, computation and local memory access patterns remain the same provided that the two candidate parents sets are of the same size. Thus, the computation of OP sets can be carried out simultaneously for each group (bucket) of CP sets of equal size. Therefore, all CP sets are first grouped by size. Then, all sets of size d from a given bucket are mapped to a d-dimensional lattice as described in Section V-A, using any 1-to-1 mapping function.

2) Scheduling: In order to maintain optimal efficiency while computing multiple F scores simultaneously, the following scheduling can be utilized. Given $p = 2^k$ processors, from the run-time analysis it follows that any number of F scores for candidate sets of size d can be computed efficiently as long as $k + \log k < d$. Thus, after grouping all CP sets by size into buckets, we proceed to compute the F scores for each bucket for which the optimality condition is met using all p processors. If the optimality condition is not met, we select k' such that $k' + \log k' < d$ and then divide $p = 2^k$ into $2^{k-k'}$ groups of processors and proceed to compute the F scores for that bucket optimally using the smaller sub-groups of processors. Since all subsets for which F scores are being computed are of equal size, we have perfect balancing within each group of $2^{k'}$ processors, as well as between the $2^{k-k'}$ different subgroups of processors. We note that both p and any sub-group of processors $2^{k'}$ are powers of 2 and thus it is guaranteed that all processors can be divided between compute tasks.

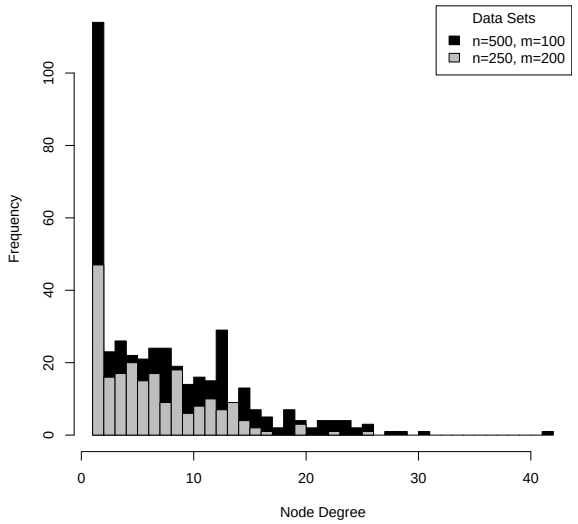

Fig. 2. Node degree distribution for CPs for datasets of sizes $n = 250$, $m = 200$, and $n = 500$, $m = 100$.

3) Run-time and Space Complexity: Consider computing a total of b F scores simultaneously, each for a CP of size d. Then, both the serial and parallel run-times are increased by a factor of b, resulting in $O\left(b \cdot mn \cdot 2^d\right)$ serially and $O\left(b \cdot mn \cdot \left(k+2^{d-k}\right)\right)$ in parallel. Thus, the parallel algorithm retains its optimality for $k + \log k < d$. The key reason for pooling F computations for nodes with CPs of the same size is to save on communication costs. By pooling together computations for b nodes, each communication size will increase by a factor of b, while the number of parallel communications remains the same.

Similarly, both the serial and parallel space complexity increase by a factor of b, and are given by $b \cdot \sqrt{\frac{2}{\pi}} \cdot \frac{1}{\sqrt{d}} \cdot 2^d$ and $b \cdot \sqrt{\frac{2}{\pi}} \cdot \frac{1}{\sqrt{d-k}} \cdot 2^{d-k}$, respectively. This results in the same serial/parallel memory consumption ratio as shown for a single variable.

VI. Resolving Highly Connected Nodes

In large networks it could be the case that some nodes would have large CP sets. For example, gene networks are usually

scale-free, and thus the node degree distribution follows the power law: $P(d) = Ad^{-\gamma}$, where $P(d)$ denotes the fraction of nodes with degree d and $A \in [2, 3]$ is a constant [5]. For such cases, we propose a parallel technique for efficient node degree reduction.

Let d_{lim} be the maximum node degree allowed. First, OP sets are computed for all variables $X_k \in \mathcal{X}$ such that $CP(X_k) < d_{lim}$. Next, node degree reduction for all remaining variables $X_i \in \mathcal{X}$ such that $CP(X_i) \geq d_{lim}$ is attempted. For each such variable X_i, all candidate parents $X_j \in CP(X_i)$ are considered. If $OP(X_j)$ has already been computed, and X_i has been selected as one of the optimal parents $X_i \in OP(X_j)$, then we deem X_j a child of X_i and remove it from the candidate parents set $CP(X_i)$, thus reducing $|CP(X_i)|$ by 1.

A. Parallel Connectivity Reduction

Consider revising the candidate parents set $CP(X_i)$ of variable X_i: $X_j \in CP(X_i)$ is removed from $CP(X_i)$ if $X_i \in OP(X_j)$. Serially, this check is performed for each CP set in each remaining bucket, for each variable within that set. We note that since each processor computes optimal parents sets for different variables, these sets are now distributed across all nodes. A trivial parallelization would require tracking where the optimal parents set resides, for each variable in every given CP set.

We perform this reduction for all variables and optimal parents sets simultaneously as follows. First, we transform all OP sets computed thus far and the remaining CP sets into pairs representation. For each $X_k \in OP(X_j)$ create a pair (X_k, X_j), where the first variable indicates the optimal parent X_k, and the child X_j is recorded as the second variable. Such pairs are marked as generated from an OP set. Further, $\forall X_j \in PC(X_i)$ we create a pair (X_i, X_j), where the candidate parent X_j is now the second variable, and the candidate child X_i is the first. Such pairs are marked as generated from a CP set. The CP sets are assigned to processors in a round-robin fashion. Every processor generates OP pairs from its local OP sets, and CP pairs from the CP sets assigned to it. Once all pairs are generated, a global parallel sort is invoked, using the first variable as primary key, and the second variable as secondary key.

After sorting, all duplicates will appear consecutively. Duplicate pairs (X_i, X_j) would appear only if the candidate child variable X_i was selected as optimal parent $X_i \in OP(X_j)$ for X_j. Thus, we remove all CP pairs for which a duplicate OP pair was available and update the remaining CP lists. The pair-representation allows for the generation of OP and CP pairs by all processors simultaneously. Next, new buckets are formed from the remaining CP sets and OP sets are computed as described in V-C.

B. Run-time and Space Complexity

Processors are assigned CP sets for CP-pair generation in a round-robin manner, with the aim of evenly distributing the number of resulting CP-pairs per processor. The total number of CP pairs is $\sum_{d=d_{lim}}^{d_{max}} b_d \cdot d$, where d_{max} is the highest node degree and b_d is the number of CP sets in a bucket with degree d. This number depends on the application and particular dataset. However, as our application shows, the total number of nodes with large connectivity is moderate (see Table I). The OP sets reside on the processor with the highest rank within its local group of $2^{k'}$ processors. For CP sets of size $d \geq k$ where $k = \log p$, the resulting OP's will reside on the processor with the highest rank $p-1$. The number of OP's is clearly bounded by the total number of variables n. The size of OP is bounded by d_{lim}. Parallel sort is performed on all OP and CP pairs. In our application we use parallel sample sort.

VII. Parallel Cycle Removal

The OP sets for all variables form a directed graph, which may have cycles. One way to globally detect all cycles in a directed graph is by exponentiating its adjacency matrix. Let $M_{n\times n} = \{m_{ij}\}$ be the adjacency matrix of the directed graph which corresponds to all optimal parents sets. If for some power $2 \leq q \leq n-1$ the trace of the matrix $M^q_{n\times n}$ is non-zero, then all non-zero elements in the main diagonal $m_{ii} \neq 0$ of $M_{n\times n}$ participate in a cycle of length q. The multiplicity of each node (the value of m_{ii}) corresponds to the number of paths which go through this node. Thus, exponentiating the adjacency matrix $M_{n\times n}$ to power q is sufficient for detecting 1) if cycles of length q are present, 2) which nodes participate in cycles of length q, and 3) how many cycles of length q pass through each of the participating nodes. If the presence of at least one cycle is detected, depth first search can be used to enumerate all cycles. Further, edges are removed following the minimal score-loss heuristic.

We exponentiate $M_{n\times n}$ in parallel, in a short-cycle-first manner, starting from $q = 2$. By eliminating cycles in increasing cycle length order, we ensure that after breaking all cycles of length q, no cycles of shorter length are present in $M_{n\times n}$. This guarantees, that we will need to exponentiate $M_{n\times n}$ at most $n-2$ times.

Once all cycles are broken, we revisit the variables whose parents sets have been modified during the cycle breaking procedure. For each such variable, we re-optimize the F score within the modified candidate parents set, as described in Section V-C.

VIII. Experimental Results

We implemented the presented algorithms in C++ and MPI. To assess the performance and applicability of the algorithms, experiments were performed on a Cray system with two 12-core AMD 'ManyCores' 2.1-GHz processors and 32 GB DDR3 1333-MHz memory per node. Each MPI process was run with 8 GB of memory. First, we analyzed the quality of the networks learned using the proposed approach. Next, to evaluate the scalability we performed experiments on a medium-size data set of $n = 500$ variables and $m = 100$ observations. Finally, we demonstrated the applicability of our

methods to larger-size problems by learning the BN structure on $1,500$ variables and 100 observations.

A. Network Quality Assessment

To evaluate the quality of the network learned using the proposed method, we compared it to a network learned using an exact learning method, i.e. one that finds a globally optimal solution.

As discussed in Section I-A, exact structure learning is NP-hard and solutions are limited to relatively small data sets. Therefore, in this analysis we use a small network of 24 genes and $1,000$ observations. We generated a synthetic gene expression data set using the well-established benchmarking package $SynTReN$ [26]. As an optimization criterion we implemented the Bayesian Dirichlet equivalence scoring function (BDe) [11], which we briefly review below.

The BDe scoring function evaluates the posterior probability of a graph N given the observed data $D_{n\times m}$. The best network is the one which maximizes the posterior probability. Let r_i denote the number of discrete states of variable X_i, and q_i be the number of possible configurations of its parent set $Pa(X_i)$ in N. Let w_j for $j = 1, \ldots, q_i$ be one of these configuration. Let also N_{ijk} be the number of instances in the data set $D_{n\times m}$, such that the variable X_i is in state k with parents in configuration j. Thus, N_{ij} is the number of instances where the parents of variable X_i are in configuration j. The BDe score is given by:

$$Score_{BDe}(N : D_{n\times m}) = \log(p\,(N))$$
$$+\sum_{i=1}^{n}\sum_{j=1}^{q_i}\log\left(\frac{\Gamma\,(\eta_{ij})}{\Gamma(N_{ij}+\eta_{ij})}\right)+\sum_{k=1}^{r_i}\log\left(\frac{\Gamma\,(N_{ijk}+\eta_{ijk})}{\Gamma\,(\eta_{ijk})}\right),$$

where $p\,(N)$ is the prior probability of structure N, η_{ijk} are the hyperparameters for the Dirichlet prior distribution of the parameters given the network structure N, and $\eta_{ij} = \sum_{k=1}^{r_i}\eta_{ijk}$. When c is an integer, the Gamma function $\Gamma(c) = \int_0^\infty e^{-u}.u^{c-1}\,\mathrm{d}u$ is given by $\Gamma(c) = (c-1)!$. The hyperparameters are computed as follows:

$$\eta_{ijk} = \eta \times p(x_{ik}, w_{ik}|G_0),$$

where $p(.|G_0)$ represents a probability distribution associated with a *prior Bayesian network* G_0 and η is a parameter representing the equivalent sample size. In our implementation of the BDe score we use the variation BDeu [8], where a uniform probability is assigned to each configuration of variable state and parent set configuration:

$$p(x_{ik}, w_{ik}|G_0) = \frac{1}{r_i q_i}.$$

We also assign uniform prior probability to all possible structures, and therefore ignore the $\log(p\,(N))$ term. The BDe assigns equal scores to graphs which equally well represent the dependencies in $D_{n\times m}$.

The proposed heuristic approach learned a network with BDe score of $-18,000.007$, while the exact algorithm learned a network with BDe score of $-17,832.3$. Thus, the heuristic algorithm produced a network with total score within 0.94% of the optimal.

TABLE I
CANDIDATE PARENTS SETS STATISTICS FOR THE TEST DATA SET OF SIZE $n = 500, m = 100$.

Data set	$D_{500\times100}$	$D_{1500\times100}$
Num. Edges	1,643	9,230
Max Node Degree	42	85
Avg Node Degree	7.9	13.9
d_{lim}	25	15
Num. CP Sets of size $\geq d_{lim}$	9	543
Num. CP Sets of size ≥ 1	415	1,170

B. Data Preparation

In order to demonstrate the scalability of our algorithm and apply it to biological problems of interest, we created a large gene expression dataset by collecting and processing microarray data from public repositories. A total of $3,546$ non-redundant Affymetrix expression profiles of the plant *Arabidopsis thaliana* generated under diverse experimental conditions were collected from the NASCArray [2], AtGenExpress [4], ArrayExpress [1], and GEO [3] repositories and pre-processed as described in [27]. This resulted in a data-matrix of $3,137$ observations and $15,596$ probe-sets, mapping to $15,495$ genes. We created a medium-scale data set consisting of 500 genes and 100 observations, and a larger data set of $1,500$ genes and 100 observations, by selecting subsets of genes and subsamples.

The values for each variable are the gene expression levels detected by microarray experiments, which are continuous. We represent gene expression using a multinomial model. After the normalization performed in [27], we discretized the data into three levels, using the average expression value of each gene across experiments as control in a similar manner as in [9]. The candidate parents (CP) were pre-computed using the parallel information-theoretic approach proposed by Zola *et al.* [27] with CP statistics summarized in Table I.

C. Scalability Analysis

To assess the scalability of the proposed approach we evaluated its performance as a function of the number of processor cores for a data set with $n = 500$ genes and $m = 100$ observations. The run-times for the different stages of the parallel framework are shown in Table II. The same data for computing optimal parents sets before and after degree reduction, as well as, the total run time are plotted in Figure 3 to show relative speedup when compared to a 4-core run. These results were generated by setting the maximal allowed node-degree to $d_{lim} = 25$.

The results show that initial estimation of the candidate parents sets using the parallel algorithm presented in [27] takes negligible time in comparison the total run-time of the subsequent parallel Bayesian structure learning algorithm. Of the multiple steps in our parallel structure learning algorithm,

TABLE II

TOTAL RUN-TIME IN SECONDS FOR THE COMPUTATION OF: CANDIDATE PARENTS (CP), OPTIMAL PARENTS SETS OF SIZE NO MORE THAN 25 (OP1), NODE DEGREE REDUCTION (DR), COMPUTATION OF REMAINING OPTIMAL PARENTS SETS (OP2), CYCLE REMOVAL (CYC), AND TOTAL RUN-TIME FOR THE PROPOSED APPROACH (TOT), FOR THE DATA SET WITH $n = 500$ GENES AND $m = 100$ OBSERVATIONS.

No. Cores	CP	OP1	DR	OP2	CYC	TOT
4	6.503	10,375	0.040	11,283	82.03	21,740
8	3.592	5,267	0.020	5,664	43.21	10,974
16	2.084	2,717	0.011	2,904	25.09	5,646
32	1.265	1,365	0.005	1,457	16.40	2,839
64	0.788	705	0.006	740	11.00	1,457
128	0.523	377	0.010	390	11.63	798
256	–	173	0.014	184	7.02	364
512	–	87	0.030	91	6.85	185
1024	–	49	0.072	50	7.42	107

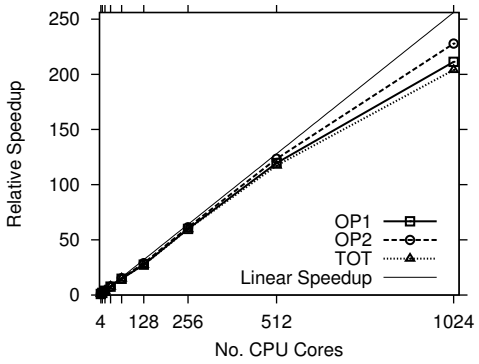

Fig. 3. Relative speedup when compared to a 4-core run for the data set with $n = 500$ genes and $m = 100$ observations, for the computation of optimal parents sets from CP sets of size no more than 25 (OP1), the remaining ones (OP2), and total run-time (TOT).

computing optimal parents using the exact algorithm is the most time consuming task. As shown in Table I, of the total 415 variables with CP sets of size 1 or larger, only 9 exceed $d_{lim} = 25$. Nevertheless, due to the exponential complexity of computing optimal parents, the time required to compute the remaining 9 large sets is on average longer than the 406 smaller sets. In this test case, the time for degree reduction is negligibly small. This is to be expected as CP pairs are created for only 9 variables, and the resulting network is sparse. However, as node degree reduction is dominated by parallel sort, we expect it to have good scaling for denser networks. Similarly, the adjacency matrix exponentiation and cycle removal is relatively fast. This would change as a function of the sparsity of the network. For the presented test data, 2 of the 9 large CPs were successfully reduced to size smaller than 25 using the connectivity reduction method proposed in VI-A. We note that the computation of optimal parents exhibits nearly perfect scaling on up to $1,024$ cores. As good scaling on up to $1,024$ cores is observed for a network of size $n = 500$, we expect that larger networks will show good scaling and can utilize much larger number of cores efficiently.

Finally, we applied the proposed method to a larger data set of $n = 1,500$ genes and $m = 100$ observations, but with a lowered d_{lim} of 15. The statistics of the candidate parents set sizes for this run are summarized in the rightmost column in Table I. Our approach computed the network in approximately $1,683$ seconds on 64 cores. Optimal parents sets for nodes whose candidate parents sets fall under d_{lim} were computed in 50 seconds. Computing optimal parents sets of larger degree nodes using the exact algorithm is the biggest contributor to the run-time at $1,338$ seconds. Cycle detection and removal accounted for an additional 295 seconds, with all other steps taking negligible run-time.

IX. CONCLUSIONS AND FUTURE WORK

In this paper, we present a parallel BN structure learning algorithm that combines the heuristic approach of estimating the potential parents set of each node in the network with an exact algorithm that refines such sets to produce the final network. To the extent that the initial estimates contain the true parents, the exact algorithm on the constrained sets should produce results identical to those obtained as if the exact algorithm is run on the whole network. We demonstrate that the resulting algorithm is scalable and fast – networks of size 500 were inferred in less than 2 minutes on 1024 cores of a Cray AMD cluster. These results indicate that much larger networks can be built using our approach.

ACKNOWLEDGMENTS

This work is supported in part by the U.S. National Science Foundation under CCF-0811840, CCF-0751157, as well as, through TeraGrid/XSEDE resources provided by TACC under TG-ASC110004. The authors would like to acknowledge Jaroslaw Zola and Maneesha Aluru for providing us with the whole-genome expression profiles from *Arabidopsis thaliana* used in our experiments. The authors thank Abhinav Sarje for carrying out part of the experiments.

REFERENCES

[1] EMBL-EBI ArrayExpress. http://www.ebi.ac.uk/microarrayas/aer/, 2009.
[2] NASC European Arabidopsis Stock Centre. http://www.arabidopsis.info/, 2009.
[3] NCBI Gene Expression Omnibus. http://www.ncbi.nlm.nih.gov/geo/, 2009.
[4] TAIR. http://www.arabidopsis.org/, 2009.

[5] A.-L. Barabasi and Z. N. Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113, February 2004.

[6] J. Cheng. Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence*, 137(1-2):43–90, May 2002.

[7] D. M. Chickering, D. Heckerman, and D. Geiger. Learning Bayesian networks is NP-hard. Technical Report MSR-TR-94-17, Microsoft Research, 1994.

[8] G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.

[9] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using Bayesian networks to analyze expression data. *Journal of Computational Biology*, 7(3):601–620, August 2000.

[10] N. Friedman, I. Nachman, and D. Pe'er. Learning bayesian network structure from massive datasets: The Sparse Candidate algorithm. In *In Proc. of Uncertainty in Artificial Intelligence*, pages 206–215, 1999.

[11] D. Heckerman, D. Geiger, and D. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

[12] W. Lam and F. Bacchus. Learning Bayesian belief networks an approach based on the MDL principle. *Computational Intelligence*, 10:269–293, 1994.

[13] A. Moore and W.-K. Wong. Optimal reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning. In T. Fawcett and N. Mishra, editors, *In ICML: Twentieth International Conference on Machine Learning*, pages 552–559. AAAI Press, August 2003.

[14] R. E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.

[15] O. Nikolova and S. Aluru. Parallel discovery of direct causal relations and markov boundaries with applications to gene networks. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 512 –521, sept. 2011.

[16] O. Nikolova, J. Zola, and S. Aluru. A parallel algorithm for exact Bayesian network inference. In *HiPC: Sixteenth International Conference on High Performance Computing*, pages 342–349, 2009.

[17] S. Ott, S. Imoto, and S. Miyano. Finding optimal models for small gene networks. In *Pacific Symposium on Biocomputing*, pages 557–567, 2004.

[18] H. Peng and C. Ding. Structure search and stability enhancement of Bayesian networks. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 621 – 624, nov. 2003.

[19] R. W. Robinson. Counting labeled acyclic digraphs. In F. Harary, editor, *New Directions in Graph Theory*. New York: Academic Press, 1973.

[20] K. Sachs, O. Perez, D. Pe'er, D. A. Lauffenburger, and G. P. Nolan. Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data. *Science*, 308(5721):523–529, April 2005.

[21] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[22] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search, 2nd Edition*, volume 1 of *MIT Press Books*. The MIT Press, June 2001.

[23] Y. Tamada, S. Imoto, H. Araki, M. Nagasaki, C. Print, D. S. Charnock-Jones, and S. Miyano. Estimating genome-wide gene networks using nonparametric Bayesian network models on massively parallel computers. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 8(3):683–697, May 2011.

[24] M. Teyssier and D. Koller. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *In UAI*, pages 584–590, 2005.

[25] I. Tsamardinos, L. Brown, and C. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65:31–78, 2006. 10.1007/s10994-006-6889-7.

[26] T. Van den Bulcke, K. Van Leemput, B. Naudts, et al. SynTReN: a generator of synthetic gene expression data for design and analysis of structure learning algorithms. *BMC Bioinformatics*, 7:43, 2006.

[27] J. Zola, M. Aluru, A. Sarje, and S. Aluru. Parallel information-theory-based construction of genome-wide gene regulatory networks. *IEEE Transactions on Parallel and Distributed Systems*, 21:1721–1733, 2010.